Regression von Nutzerbewertungen mit supervised Learning, am Beispiel osu! Beatmaps

von Robin Korn und David Deml

Repository: https://git-kik.hs-ansbach.de/2023_studentenprojekte/osu

Gliederung:

1. Ziel des Projekts

Ziel des gesamten Projektes ist, die Nutzerbewertungen von osu! Maps anhand von verschiedensten Featuren der Map selbst vorherzusagen, also eine Regression. In diesem Fall werden Websitedaten, Mapdaten und Audiodaten der Map betrachtet, um die Nutzerbewertung vorherzusagen.

2. osu!

2.1 Was ist osu!?

osu! ist ein kostenloses Rhythmusspiel, das hauptsächlich von Dean “peppy” Herbert entwickelt und primär an einem PC gespielt wird. Die erste veröffentlichte Version wurde am 16. September 2007 veröffentlicht und wird dauerhaft vom “osu! development team” weiterentwickelt. Das “osu! development team” arbeitet sowohl an osu! als auch an der neuen Version des Spiels osu!lazer, welches demnächst veröffentlicht werden soll und derzeit (Stand 09.05.2023) in der Beta läuft. Durch osu!lazer wird es in Zukunft auch auf mobilen Endgeräten möglich sein, das Spiel zu spielen.
Peppy wurde von einem Spiel namens “Osu! Tatakae! Ouendan” von “iNiS'” inspiriert, bis er sich letztendlich entschloss, sein eigenes Spiel zu programmieren.
osu! bietet vier verschiedene Spielmodi, osu!, osu!mania, osu!taiko und osu!catch, welche alle auf einem anderen Prinzip basieren. Für dieses Projekt ist jedoch nur der klassische osu! Modus von Gebrauch.
Hauptsächlich wird osu! von den Spielern selbst betrieben, alle Maps, alle Regeln und Moderatoren und sogar Teile des “osu development teams” bestehen aus Spielern, beziehungsweise von Spielern geschaffenen Dingen, die von einem Komitee geleitet werden.
Das Spielprinzip basiert auf einem Rhythmus Prinzip. Es gibt genügend Maps von vielen Songs, in denen Objekte auf den Beat des Songs platziert werden. Der Spieler muss diese Objekte im richtigen Zeitpunkt erwischen, um den maximalen Score zu erzielen.

2.2 Was ist eine osu! Beatmap?

Eine osu! Beatmap befindet sich lokal auf dem Rechner des Spielers in einem Ordner.
In diesem Ordner gibt es verschiedene Dateien, die die Map in Zusammenarbeit darstellen. Natürlich wird eine Audiodatei benötigt um den Song wiederzugeben, diese ist normalerweise in einem mp3 Format und heißt somit auch “audio.mp3”. Die Objekte der Map werden in einer .osu Datei gespeichert, diese ist eine vom Entwickler eigene Datei, die die Informationen in einer “menschlich lesbaren” Form darstellt. In

dieser .osu Datei findet man alle Informationen zur Map, welche Objekte an welcher Stelle sind, die Namen der wichtigen anderen Dateien, die BPM und Zeitpunkte, als auch Informationen zum Song und Ersteller der Map. Jede Map benötigt auch ein Hintergrundbild, das im Spiel dann dargestellt wird, meistens ist diese als .jpg oder .png im Ordner vorhanden. Diese 3 Dateien reichen schon aus, um eine Map zu erstellen und zu spielen. osu! bietet viele Möglichkeiten, um die Map zu erweitern durch mehrere Audiodateien, die die Map verschönern oder ein Storyboard, welches ein künstlich animiertes Hintergrund-Video im Spiel darstellt. Natürlich können auch mehrere .osu Dateien für mehrere Maps in einem Set existieren und somit auch mehrere Audiodateien, die verschiedenen Songs in einem Set entsprechen.

2.3 Welche Form kann eine osu! Beatmap haben?

Eine osu! Beatmap hat so gesehen keine Begrenzung in der Form. Es ist möglich jede Audiodatei in eine Map zu verwandeln, unabhängig davon ob die Audiodatei überhaupt ein Song ist. Es gibt aber nicht nur unbegrenzt viele möglichen Audiodaten sondern auch Mapstile die sich Ersteller solcher Maps einfallen haben lassen. Über die fast 15 Jahre seit der Veröffentlichung von osu! entstanden viele solcher Mapstile. Ein paar dieser Stile sind zum Beispiel, Old-Style, Consistency-Style, Stream-Style, Tech-Style, Gimmick-Style, Reading-Style, Movement-Style als auch tausende andere. Jeder Stil unterscheidet sich natürlich auch anhand der jeweiligen Audiodatei und bietet viel Variation, daher wird es auch oft Ausreißer in den Daten geben, da so gesehen keine gleichen, aber ähnlichen, Maps existieren.

3. Auswahl der Trainingsdaten

Die Auswahl der Trainingsdaten spielt eine wichtige Rolle bei jedem Machine Learning Modell, auch bei diesem Beispiel muss auf ein aussagekräftiges Datenset geachtet werden um so gut wie möglich eine Vorhersage zu treffen. Da sich die Nutzerbewertungen normalerweise nach etwas Zeit zwischen 8 und 9.5 einpendeln gibt es ausreichend Maps mit solchen Bewertungen. Schwierig ist es Maps unter 8 und über 9.5 zu finden. Durch Filter berücksichtigen wir auch die am besten bewerteten Maps bzw. die am schlechtesten bewerteten Maps. Es gibt keine aussagekräftigen Maps die eine Bewertung unter 5 haben. Der untere Bereich von “schlechten” Maps liegt also zwischen 5 und 8, während der obere Bereich von extrem beliebten Maps zwischen 9.5 und circa 9.9 liegt. Eine Bewertung von 10 ist immer unwahrscheinlich, durch Manipulation der Nutzer selbst. Für dieses Beispiel haben wir uns entschieden um die 1.500 Maps als Trainingsdaten zu verwenden, diese resultieren in etwa bei circa 4.500 samples aufgrund der Verteilung von Maps in einem Mapset. Jeweils circa

200 der oberen und unteren Hälfte dieser Maps sind Extrembeispiele, um die Grenzen mit einzubeziehen.

4. Verwendung der Daten

4.1 Welche Website-Daten werden verwendet?

Für das auslesen der Website-Daten werden API Requests über die ossapi Library in Python benutzt.
Natürlich muss von den Websitedaten der Wahrheitswert, die Nutzerbewertung zurückgeliefert werden. Abgesehen davon muss man die Website-Daten in zwei Arten von Daten unterteilen. Zum einen gibt es die “rohen” Websitedaten und zum anderen die Map-Daten, welche ebenfalls auf der Website zu finden sind. Die “rohen” Website Daten beinhalten Werte, welche nichts über die Map selbst aussagen, Feature wie, wie oft eine Map gespielt wurde, der Playcount, oder wie oft eine Map favorisiert wurde, der Favouritecount. Map-Daten sind Daten, die sich auf die Map selbst beziehen, sowas wie die Schwierigkeitseinstellungen.
Alle verwendeten rohen Websitedaten: rating, diffcount, bpm, ranked_date, favourite_percentage, favourite_count, playcount, passcount, successrate
Als auch die Map-Daten: difficulty, language, genre, creator, artist, total_length, approach_rate, circle_size, hp, overall_difficulty, aim_difficulty, approach_rate_diff, overall_difficulty, speed_note_count, top_map_player, pp_value
Weitere Informationen zu jedem Wert stehen im Repository selbst in der README.md.
Wichtig hier zu beachten ist jedoch, das Websitedaten teilweise auch als Mapdaten angesehen werden können und dies eine Frage der Definition ist, dazu in Punkt 4.2 mehr.

4.2 Welche Map-Daten werden verwendet?

Für das auslesen der Map-Daten verwenden wir osuparser, jedoch nicht als Library sondern als lokale Dateien im Repository, da es beim installieren dieser “Library” Probleme gab. Teilweise kam es dann auch zu Fehlern weshalb Teile dieser Library manuell von uns angepasst werden mussten.
Wie schon erwähnt gibt es teilweise Map-Daten auch auf der Website zu finden, dazu zählen Schwierigkeitseinstellungen als auch genre, creator, artist, language und total_length. Alle diese Feature findet man theoretisch auch in der .osu File, jedoch hat es sich angeboten, diese Feature von der Website zu ziehen und zu verwenden. Im Folgenden zählen diese Feature dann auch als Website Feature obwohl diese eigentlich Map Feature sind.

Andere Map Feature die mit Hilfe der osuparser Library berechnet worden sind, sind benötigte Geschwindigkeiten und Beschleunigungen zwischen Objekten und Patterns, als auch Winkel, Distanzen, Zeitdifferenzen und eine eigene Formel für die “Rhythm Complexity” die dem Schwierigkeitsgrad der Website Feature in gewisser Weiße ähnelt.
Die genaue Liste (ohne besagte Website Feature die als Map Feature zählen könnten): max_v, mean_v, max_x, min_t, mean_x, max_a, mean_a, max_rhythm_complexity, max_angle, mean_angle, min_angle.

4.3 Welche Audio-Daten werden verwendet?

Für das berechnen der Audio-Daten haben wir eine Fourier-Transformation mit Hilfe der librosa Library in Python gemacht.
Unsere Audio-Daten bestehen aus zwei Teilen, eine normale Fourier Transformation Analyse mit Aufteilung in Hz Bereiche, und eine Analyse der Werte in zehn Intervallen.
Für die normale Fourier Transformation haben wir uns entschieden, die wichtigsten Hz Bereiche aufzuteilen, um genauere Informationen über den Song und dessen Struktur zu erhalten. Nachfolgend eine Liste mit den Hertz Bereichen.
40-80 Hz, 80-250 Hz, 250-600 Hz, 600-4000 Hz, 4000-6000 Hz, 6000-8000 Hz, 8000-20000 Hz
Jeder dieser Bereiche hat seine eigenen Eigenschaften, während Gesang im Bereich 4000-8000 Hz ist, ist zum Beispiel Bass zwischen 40-250 Hz und hohe Töne wie das Becken eines Schlagzeugs zwischen 8000-20000 Hz.
In diesen Bereichen schauen wir uns jeweils den Durchschnitt der aufkommenden Frequenzen an was letztendlich in diesen Featuren endet:
40-80 Hz Avg, 80-250 Hz Avg, 250-600 Hz Avg, 600-4000 Hz Avg, 4000-6000 Hz Avg, 6000-8000 Hz Avg, 8000-20000 Hz Avg
Beim anderen Teil der Audio Feature beziehen wir uns auf Intervalle des Songs, der Song wird auf genau 10 gleich lange Intervalle aufgeteilt, hier ist jedoch zu beachten, dass dies die Werte und deren Aussagekraft etwas verfälscht, da mit variabler Song Länge und dementsprechend variabler Intervalllänge, andere Werte entstehen, beziehungsweise Werte, die sich eventuell ähneln, obwohl komplett verschiedene Sachen im Song passiert sind. Trotzdem haben wir Maximas, Minimas und den Durchschnitt von diesen Intervallen benutzt, für Maximas und Minimas verwenden wir nicht die absoluten Maximas und Minimas, sondern die 97% höchsten Werte und 3% niedrigsten Werte, um Ausreißer in den Daten zu vermeiden.

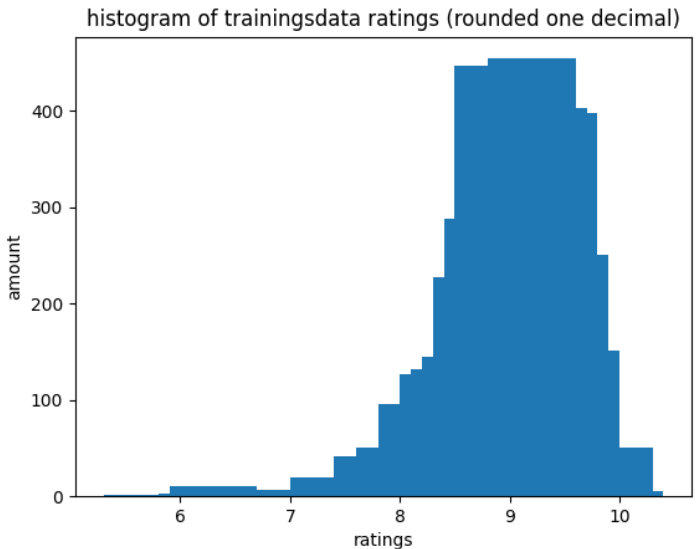
Dadurch entstehen folgende zusätzliche Audio-Feature:
Avg of IV 1, Avg of IV 2, Avg of IV 3, Avg of IV 4, Avg of IV 5, Avg of IV 6, Avg of IV 7, Avg of IV 8, Avg of IV 9, Avg of IV 10, 3 Percent of IV 1, 3 Percent of IV 2, 3 Percent of IV 3, 3 Percent of IV 4, 3 Percent of IV 5, 3 Percent of IV 6, 3 Percent of IV 7, 3 Percent of IV 8, 3 Percent of IV 9, 3 Percent of IV 10, 97 Percent of IV 1, 97 Percent of IV 2, 97 Percent of IV 3, 97 Percent of IV 4, 97 Percent of IV 5, 97 Percent of IV 6, 97 Percent of IV 7, 97 Percent of IV 8, 97 Percent of IV 9, 97 Percent of IV 10.

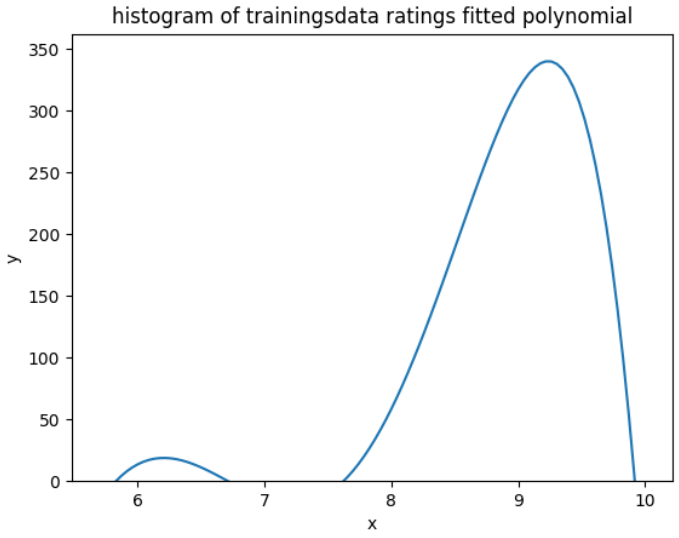
5. Auswahl Machine Learning Modell

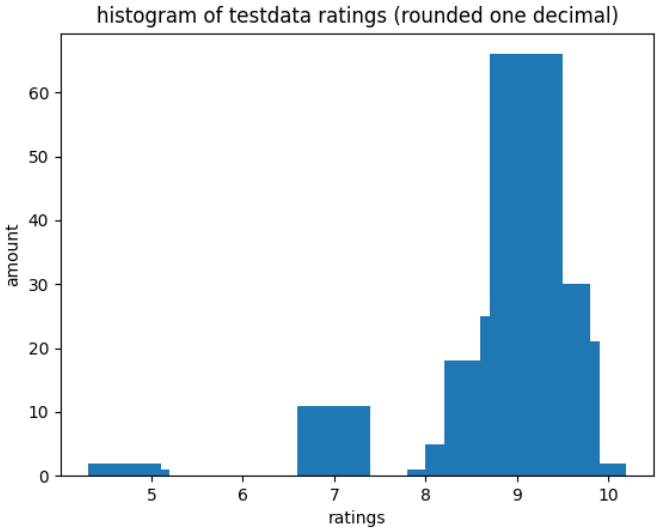
Für das Projekt haben wir uns für den RandomForestRegressor von sklearn entschieden. Dieser eignet sich gut für Regressionen mit vielen Samples, Features und nicht linearen Zusammenhängen zwischen den Features.
Neben dem RandomForestRegressor stand noch der DecisionTreeRegressor in Betracht. Jedoch ist beim Testen aufgefallen, dass dieser schlechter als der RandomForestRegressor abschneidet, wenn man unseren Fall betrachtet beziehungsweise die Anzahl der Daten und Featuren.
Andere Möglichkeiten wären zum Beispiel Support Vector Regression (SVR), Lineare Regression, Gradient Boosting Regression und viele mehr.
Wichtig zu erwähnen ist, dass der RandomForestRegressor durchaus nicht die beste Wahl ist, aber eine gute. Natürlich gibt es immer eine bessere Lösung, aber um diese herauszufinden, braucht man viel Kapazität und Zeit, die leider im Rahmen dieser Arbeit gefehlt haben.

6. Verteilung der Datensätze

Nachfolgend drei Bilder, die die Verteilung der Test- und Trainingsdaten darstellen. Also die x-Achse stellt die Nutzerbewertung dar und die y-Achse die Anzahl des jeweiligen Ratings. Eine dieser Grafiken veranschaulicht die Verteilung der Trainingsdaten, um darzustellen, dass diese einer Gaußverteilung ähneln, mit kleinen Ausreißern.







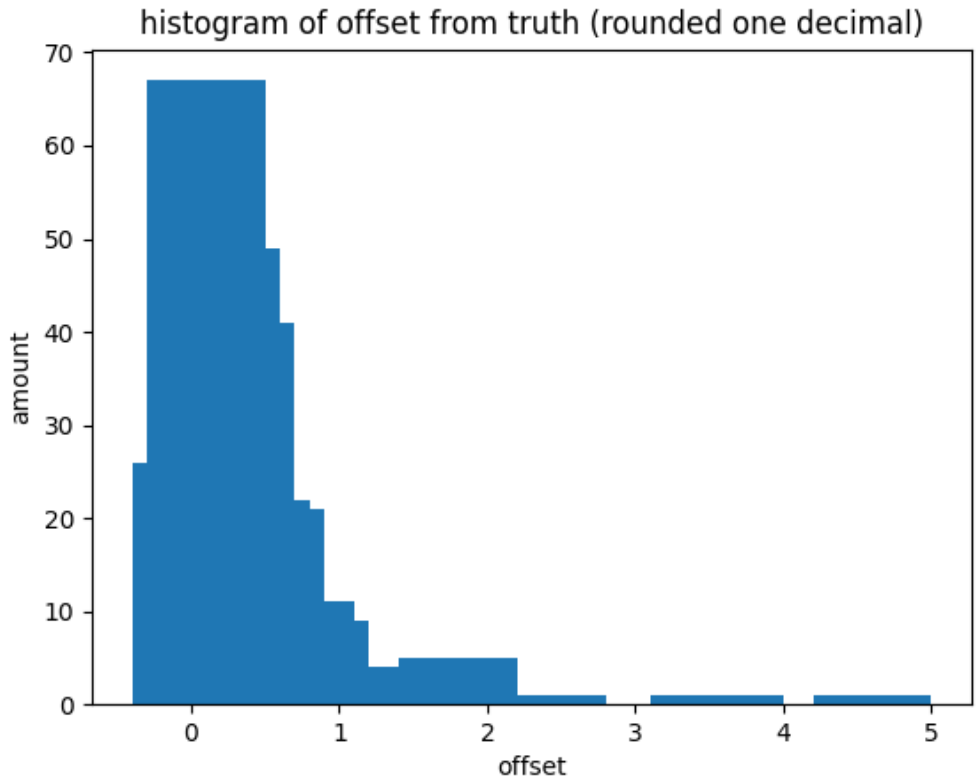
7. Auswertung der Feature und des Modells

Wichtig zu erwähnen ist das dieses Projekt standardmäßig keine Anwendung in der Realität hat, da einige Feature theoretisch erst nach dem Ranked Status einer Beatmap verfügbar sind. Wir haben dies einmal standardmäßig ausgewertet und ein zweites Mal, jedoch ohne die besagten Feature, um einen Realfall darzustellen.

Die Top fünf Feature Importances im normalen Fall sind wie folgt:

1. ranked_date: 8.04% des Ergebnisses
2. playcount: 6.28%
3. favourite_count: 5.25%
4. passcount: 4.69%
5. total_lengths: 3.43%

Die allgemeine Verteilung auf die verschiedenen Bereiche ist 43.4% Website Feature, 4.1% Map Feature und 47.5% Audiodaten, hier jedoch wieder Definition abhängig von Website- und Map Featuren.
Die ersten vier dieser fünf Feature sind Feature, die wie schon erwähnt, erst nach dem Ranked Status auslesbar sind. In diesem Fall hat der RandomForestRegressor einen mean squared error (MSE) von 0.35, also weicht dieser beim benutzen circa 0.35 vom Wahrheitswert ab. Dies kann man anhand der folgenden Grafik auch interpretieren.



Circa 98% der Predictions liegen im Bereich zwischen 0 und 1, 2% der Predictions sind dann die Ausreißer, welche noch nicht gut erkannt werden können. Lösungsvorschläge dazu dann im Punkt 9.
Im Realfall kann man die oben genannten Feature nicht wissen, weshalb wir eine zweite Auswertung gemacht haben, bei denen nur Features bewertet werden, die im Realfall bekannt wären.
Die neue Verteilung schaut so aus:

1. BPM: 8.95% des Ergebnisses
2. total_lengths: 7.97%
3. 8000-20000 Hz: 5.96%
4. 4000-6000 Hz: 5.95%
5. speed_note_count: 5.87%

Die allgemeine Verteilung auf die verschiedenen Bereiche ist in diesem Fall 52.6% Website Feature, 14.6% Map Feature´und 32.8% Audio Feature.
Der mean squared error bleibt gleich bei 0.35 auch wenn man die Feature entfernen würde.
osu! Mapper könnten dies also durchaus verwenden, um ihre Maps bewerten zu lassen.

8. Verwendung des Repositorys

Die genaue Verwendung des Repositorys steht in der README.md des Projekts und ist leider etwas aufwändig, da osu! nicht gerade für Zugänglichkeit bekannt ist.
Im Groben und Ganzen muss das Repository erstmal gecloned und geforkt werden, um es lokal auf dem Rechner zu haben.
Dann müssen zwei Ordner im Repository erstellt werden mit den Namen “data” und “input”, diese werden für Trainings und Testdaten verwendet.
Anschließend braucht man einen osu! Account, welcher nur erstellt werden kann, nachdem man osu! auf dem PC installiert hat.
Sobald dieser erstellt wurde, muss man auf der Website eine OAuth Application erstellen dazu muss man oben rechts auf das eigene Profilbild klicken, dann zu Settings (Einstellungen) und anschließend auf “New OAuth Application” klicken. Man wird dann nach einem Namen für die Application gefragt. Dieser kann natürlich beliebig gewählt werden. Anschließend muss man nur auf “Register application” klicken.
Nun muss eine .env Datei im Repository erstellt werden welche benutzt wird, um die lokalen Variablen bzw. die Application zu laden. Das vorgegebene Muster muss so aussehen:
client_id=’die client id’
client_secret=”das client secret’
Nun müssen alle fehlenden Librarys installiert werden mit “pip install -r requirements.txt”, falls es zu Fehlern kommt, kann man versuchen, mit anderen Versionen der Librarys zu arbeiten.
Nun müssen alle gedownloadeten Maps in jeweils den “data” und “input” Ordner gezogen werden. Wichtig hierbei ist, dass jede Map den Ranked Status hat.
Jetzt kann “preprocess.py” gestartet werden, welches zwei verschiedene csvs erstellt.
Anschließend “main.py” ausführen und man kriegt die Ergebnisse des RandomForestRegressors

9. Probleme und Lösungsvorschläge

Jede Map in einem Set erhält die gleichen Bewertungen auf der Website. Heißt, wenn es sowohl gute als auch schlechte Maps in einem Set gibt, wird das Ergebnis der einzelnen Maps verfälscht. Eine Lösung hierfür gibt es noch nicht.
Die Intervalle bei den Audio Featuren variieren je nach Songlänge, was die Werte etwas verfälscht. Eine Lösung hierfür wäre, eine fest gegebene Intervall Länge gleichmäßig über die Audio zu verteilen, jedoch so, dass es immer gleich viele Intervalle mit derselben Länge gibt.
Wie schon erwähnt, müssen bestimmte Feature herausgenommen werden, um den Realfall zu betrachten. Hierfür kann man in “main.py” eine Zeile ändern, um diesen zu aktivieren.
Ausreißer im niedrigen Rating-Bereich werden schlecht bis extrem schlecht erkannt.
Es gibt jedoch in den derzeitigen Daten wahrscheinlich keine Erklärung, da Maps teilweise aus unerklärlichen Gründen “Hass” abbekommen, was das echte Rating der Map dann verfälscht. Eine Lösung hierfür wäre ein Keyword Filter in den Kommentaren einer Map zu machen, um nach Schlagwörtern wie “good” oder “nice” für gute Maps beziehungsweise “bad” oder “terrible” für schlechte Maps zu suchen.

10. Fun Facts über das Projekt

Circa 1700 Zeilen selbst geschriebener Code über 13 Dateien
Ungefähr 130 Commits über die 4 Monate Arbeitszeit
11,36 Gigabytes an osu! Maps wurden verarbeitet (2451 Ordner, 41747 Dateien)
12 Stunden Laufzeit für die finale Verarbeitung
4 PC Crashes durch fehlerhaften Code
20 Code Crashes während der Verarbeitung nach x Stunden
Und circa 100 Stunden Arbeit über die 4 Monate verteilt

11. Letzten Worte

Das Projekt war durchaus ein Erfolg und hat viele Einblicke in den Workflow von einem Programm gegeben. Wir konnten viel durch das Projekt lernen, auch wenn es teilweise schmerzhaft beigebracht werden musste. Wir möchten uns bei Professor Geißelsöder bedanken, dass er dieses Projekt betreut hat und alle Fragen unsererseits gerne beantwortet hat, als auch bei Fynn Madrian aus dem 4 KIK Semester (Stand SS 2023), der uns beim erstmaligen aufsetzen des osuparsers geholfen hat und uns Vorschläge für die Audio Feature gegeben hat.